

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**Task-Sensitive Methods and Systems for Displaying
Command Sets**

Inventor(s):
Eric Rockey
Shannon Talbott
Gavin Kelly
Nancy Jacobs
Mike Hopcroft
Daniel Westreich

ATTORNEY'S DOCKET NO. MS1-562US

001290" 9806560

56
A^c

007290-9806560

1 RELATED APPLICATIONS

2 The following patent applications are related to the present application, are
3 assigned to the assignee of this patent application, and are expressly incorporated
4 by reference herein:

- 5 • U.S. Patent Application Serial No. _____, entitled "Single
6 Window Navigation Methods and Systems", bearing attorney docket
7 number MS1-560us, and filed on the same date as this patent
8 application;
- 9 • U.S. Patent Application Serial No. _____, entitled "Methods
10 and Systems of Providing Information to Computer Users", bearing
11 attorney docket number MS1-557us, and filed on the same date as
12 this patent application;
- 13 • U.S. Patent Application Serial No. _____, entitled "Methods,
14 Systems, Architectures and Data Structures For Delivering Software
15 via a Network", bearing attorney docket number MS1-559us, and
16 filed on the same date as this patent application;
- 17 • U.S. Patent Application Serial No. _____, entitled "Network-
18 based Software Extensions", bearing attorney docket number MS1-
19 563us, and filed on the same date as this patent application;
- 20 • U.S. Patent Application Serial No. _____, entitled
21 "Authoring Arbitrary XML Documents Using DHTML and XSLT",
22 bearing attorney docket number MS1-583us, and filed on the same
23 date as this patent application;
- 24 • U.S. Patent Application Serial No. _____, entitled
25 "Architectures For And Methods Of Providing Network-based
Software Extensions", bearing attorney docket number MS1-586us,
and filed on the same date as this patent application.

19 TECHNICAL FIELD

20 This invention relates generally to methods and systems that expose
21 commands in software application programs.
22
23
24
25

BACKGROUND

Typically, application programs contain command sets that include individual commands that can be used by a user when working in a particular application program. These commands are specific to the purpose of the application program. For example, a word processing application program will typically include a command set that can be used to manipulate the text and/or format of a document. These command sets, however, are not always as easy to use as one would like. This situation can be complicated when a user is not familiar with the command set of an application program that they are currently using.

Current problems with application program command sets include that they can be difficult to use or browse because of the large number of commands that can be included in a command set, and that they often times can temporarily obscure a document when a user attempts to use them. In addition, command sets are typically presented in a manner that is not related to the tasks in which the user might be engaged.

With respect to the browsing difficulty of command sets, consider the following. Command sets can typically contain many different commands that are available for use. Since, in a typical user display, there is only a limited amount of space to present information without undesirably obscuring a work area, it logically follows that not all commands can be displayed at all times for the user. To address this problem, solutions have included providing a static tool bar that can expose some commands and contain a menu structure that can be browsed by the user. Consider, for example, Fig. 1 which shows an exemplary user display 10 that includes a tool bar 12 that includes a menu structure 14 and a collection of

commands 16. The menu structure 14 includes individual entries, e.g. “File”, “Edit”, “View”, “Insert”, etc. Each of these entries is associated with a drop down menu that contains a collection of individual commands that are logically related to their entry. For example, for the “File” entry, individual drop down menu-accessible commands include “new”, “open”, “close”, “print”, “print preview” and additional commands that are accessible via an “options” selection that further extends the drop down menu to display the additional commands. Each of the top line menu entries in the menu structure 14 can be associated with a drop down menu. Also, some of the commands are gathered into broad groups (such as the Font Formatting dialog) and the user needs to know what Font Formatting is, in order to find the commands in this group. Needless to say, the number of available commands can often times be quite numerous so that browsing through them is not an easy task. In addition, an inherent inefficiency with this approach is that many if not most of the displayed commands will have little or nothing to do with what the user is currently looking for. Often, in fact, many of the commands are grayed out and disabled because they are not relevant to the current task. Regardless, they are exposed to the full subset of commands.

Consider how this problem is exacerbated when a user is only moderately familiar or not familiar at all with an application program. Having to figure out and navigate through an extensive set of commands that are not explained (except for perhaps a “Help” dialog) can make the user’s experience difficult and time consuming. In addition, many application programs are physically or logically tapped out as far as including additional commands. Specifically, in many application programs there are simply so many commands in the command set that including more commands would require the menu structure to include more

1 “options” buttons that, in turn, would present more and more commands, thus
2 requiring a user to physically navigate through more commands.

3 Additionally, many application programs display their command sets in a
4 manner that can obscure the work area for the user. For example, consider Fig. 2
5 which shows a “Font” dialog box 18 that is presented in the middle of the user’s
6 work area. To get to this dialog box, the user had to click on the “Format” menu
7 entry in menu structure 14, and then select the “Font” command within the drop
8 down menu that was presented. As a result the illustrated dialog box 18 is
9 displayed. Although this is helpful for enabling a user to see an extensive list of
10 commands, it is not optimal for a couple of different reasons. First, the user’s
11 document is partially obscured by the dialog box 18. This is undesirable because
12 the user may wish to keep the work area in view. In addition, in order to work
13 within the dialog box, the user has to quit working within their document. Thus,
14 the dialog box is referred to as having “mode” or being “modal”, meaning that a
15 user must enter into a particular mode that renders them unable to work within
16 their document in order to work within the dialog box. Second, and perhaps more
17 important, the user’s command selection is not implemented immediately, and,
18 even if it were, the document is obscured by the dialog box. Specifically, in order
19 to have a command implemented, e.g. a “strikethrough” command, the user must
20 select text in the document that they are working on and pull up the dialog box.
21 Only after they click on the “strikethrough” box and on the “OK” box is the
22 command implemented (this is somewhat related to the mode aspect mentioned
23 above). In addition, if a user desires to implement a command on multiple
24 different portions of text, they must separately select each text portion and apply
25 the command for each selected portion of text. That is, for each portion of text,

they must separately and individually pull up the appropriate dialog box to apply the command. This is not optimal.

Consider also the collection of commands 16. Many application programs provide such a feature where frequently used commands are displayed in a manner in which they can be quickly clicked on by a user and applied within the context of the application program. These commands are often presented as so-called “modeless” commands (as contrasted with “modal” commands) because they can be used while still working within a document. That is, in the illustrated example, individual commands from the collection include “bold”, “italics”, and “underline” commands. Yet, even though the goal of displaying these frequently-used commands is directed to improving user efficiency, this attempt falls short of the mark for the following reason. Even though the commands that are displayed might be considered as those that are most frequently used, their use occurrence may constitute only a very small portion of a user session, if at all. To this extent, having the commands displayed when they are not being used is wasteful in that valuable display real estate is consumed. This is a direct manifestation of the fact that the displayed commands have nothing to do with the specific context of the user. Yes--the user might in the course of their computing session have the need to use a particular command, but until that command is specifically needed by the user, its display bears no logical relation to the user’s computing context.

Accordingly, this invention arose out of concerns associated with providing improved methods and systems for presenting command sets to users. Specifically, the invention arose out of concerns associated with providing methods and systems of presenting commands in a task-sensitive manner, which assist in using physical screen space in a more efficient manner.

the incorporated functionalities can come with its own collection of automatically displayable context blocks.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is an illustration of an exemplary user display in accordance with the prior art.

Fig. 2 is an illustration of an exemplary user display that includes a dialog box in accordance with the prior art.

Fig. 3 is a high level block diagram of an exemplary computer system that can be utilized to implement various inventive embodiments.

Fig. 4 is a flow diagram that describes steps in a method in accordance with one described embodiment.

Fig. 5 is a diagram of an exemplary user interface in accordance with one described embodiment.

Fig. 6 is a diagram of an exemplary context block in accordance with one described embodiment.

Fig. 7 is a diagram of an exemplary context pane in accordance with one described embodiment.

Fig. 8 is a flow diagram that describes steps in a method in accordance with one described embodiment.

Fig. 9 is a diagram of a table in accordance with one described embodiment.

Fig. 10 is a diagram of a tree structure in accordance with one described embodiment.

1 Fig. 11 is a flow diagram that describes steps in a method in accordance
2 with one described embodiment.

3 Fig. 12 is a diagram of a user interface in accordance with one described
4 embodiment.

5 Fig. 13 is a diagram of a user interface in accordance with one described
6 embodiment that illustrates an exemplary functionality.

7 Fig. 14 is a diagram of a user interface in accordance with one described
8 embodiment that illustrates an exemplary functionality.

9 Fig. 15 is a flow diagram that describes steps in a method in accordance
10 with one described embodiment.

11 12 **DETAILED DESCRIPTION**

13 **Overview**

14 The methods and systems described below present commands to a user
15 within a software application program by determining the user's context within the
16 application program and automatically presenting, in a user interface, context-
17 sensitive commands that pertain to the user's current context. When the user's
18 context changes, the context-sensitive commands can be automatically removed
19 from the user interface.

20 21 **Exemplary Computer System**

22 Fig. 3 shows an exemplary computer system that can be utilized to
23 implement the embodiment described herein. Computer 130 includes one or more
24 processors or processing units 132, a system memory 134, and a bus 136 that
25 couples various system components including the system memory 134 to

1 operating system 158, one or more application programs 160, other program
2 modules 162, and program data 164. A user may enter commands and
3 information into computer 130 through input devices such as a keyboard 166 and a
4 pointing device 168. Other input devices (not shown) may include a microphone,
5 joystick, game pad, satellite dish, scanner, or the like. These and other input
6 devices are connected to the processing unit 132 through an interface 170 that is
7 coupled to the bus 136. A monitor 172 or other type of display device is also
8 connected to the bus 136 via an interface, such as a video adapter 174. In addition
9 to the monitor, personal computers typically include other peripheral output
10 devices (not shown) such as speakers and printers.

11 Computer 130 commonly operates in a networked environment using
12 logical connections to one or more remote computers, such as a remote computer
13 176. The remote computer 176 may be another personal computer, a server, a
14 router, a network PC, a peer device or other common network node, and typically
15 includes many or all of the elements described above relative to computer 130,
16 although only a memory storage device 178 has been illustrated in Fig. 3. The
17 logical connections depicted in Fig. 3 include a local area network (LAN) 180 and
18 a wide area network (WAN) 182. Such networking environments are
19 commonplace in offices, enterprise-wide computer networks, intranets, and the
20 Internet.

21 When used in a LAN networking environment, computer 130 is connected
22 to the local network 180 through a network interface or adapter 184. When used
23 in a WAN networking environment, computer 130 typically includes a modem 186
24 or other means for establishing communications over the wide area network 182,
25 such as the Internet. The modem 186, which may be internal or external, is

connected to the bus 136 via a serial port interface 156. In a networked environment, program modules depicted relative to the personal computer 130, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Generally, the data processors of computer 130 are programmed by means of instructions stored at different times in the various computer-readable storage media of the computer. Programs and operating systems are typically distributed, for example, on floppy disks or CD-ROMs. From there, they are installed or loaded into the secondary memory of a computer. At execution, they are loaded at least partially into the computer's primary electronic memory. The invention described herein includes these and other various types of computer-readable storage media when such media contain instructions or programs for implementing the steps described below in conjunction with a microprocessor or other data processor. The invention also includes the computer itself when programmed according to the methods and techniques described below.

For purposes of illustration, programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computer, and are executed by the data processor(s) of the computer.

Context Sensitive Commands

In the described embodiment, command sets that include one or more individual commands are automatically presented to a user depending on the

1 user's context. Specifically, depending on the type of action the user has taken,
2 commands that are specific to that action will appear automatically thus obviating
3 the need for the user to hunt through a menu structure to find commands of
4 interest. This improves upon past approaches, which always presented top level
5 commands, even when they were not needed by the user. This is also
6 advantageous from the standpoint of assisting users who are unfamiliar with a
7 particular software application. In the past, these users would have to hunt
8 through an unfamiliar menu structure to find commands that may or may not be
9 pertinent to an action that the user desired to take. Users also had to know the
10 names of the functionality in order to find the tools (e.g. the user needed to know
11 what a "table" was to know that there are tools for tables in an appropriate menu).
12 In the present case, contextually-appropriate commands are automatically
13 presented and removed in an interface so that a user need not worry about finding
14 appropriate commands. That is, the described embodiment maintains an invariant
15 that contextually applicable commands are visible and other non-applicable
16 commands are hidden from the user.

17 As an example, consider the following: A user is working in a word
18 processing application and is in the process of preparing a document. The user
19 selects, with their cursor, a portion of the text that they believe to be spelled
20 incorrectly. Instead of having to go to a tool bar menu at the top of the document
21 and pull down one or more other menus to find the spell checking feature, a spell
22 checking context block automatically appears in an interface adjacent the
23 document. The user can then correct the incorrectly spelled word using the spell
24 checking context block. Once the word is corrected and the user's context is not
25 longer associated with an incorrectly spelled word, the spell checking context

block automatically disappears. As the user's context changes within their document, so too do the sets of automatically presented and removed commands. Consider further that the user has included a table in their document and that they wish to manipulate the table or its contents with table specific commands. In the past, the user would typically have to pull down a table menu entry and then select from one or more commands, some of which might present a dialog box that would obscure the user's document. In the present example, a user would simply select the table by placing the cursor inside of the table to have table-specific commands that are contextually accurate and appropriate automatically displayed in a dedicated space. Thus, a user need not hunt through a large menu structure to find commands that are appropriate for use. Here, contextually proper commands are automatically presented for the user. As the user's context changes, so too do the displayed command sets.

Fig. 4 is a flow diagram that describes steps in a method in accordance with the described embodiment. The illustrated method can be implemented in any suitable hardware, software, firmware, or combination thereof. In the present example, the method is implement in software that is executing on a user's computer.

At step 400 the method starts and step 402 then determines whether the user's current context has changed. The user's current context relates to various tasks that the user is attempting to accomplish. Context can be determined from almost any detectable state in a program. In the above example, two exemplary tasks included correcting an incorrectly-spelled word and manipulating the contents of a table. The context associated with each of these tasks can be determined from such things as the type of document a user is working in (i.e.

designed, in a particular implementation, to work with applications whose functionalities are primarily oriented around interacting with documents. The context UI container 502 does not preclude other UI constructs in the application, and indeed this design assumes some additional UI such as a toolbar with commands on it such as would be displayed in global toolbar area 506. Such commands can include, without limitation, an address well for a web browser, a "Create New" button for creating a new document type, or a button that brings up "search" or "help".

The context UI container 502 is designed, in this example, to lay to the left of the document area 504 in the application. It could, however, be situated in any suitable location. The context UI container 502 contains two types of objects that are utilized to display context-sensitive commands to a user. A first type of object is a context block, exemplary ones of which are shown at 508a-c. Context blocks are essentially context-based palettes with command "shortcuts", giving the user top-level tools for the current user contexts. In the present example, there are context blocks for editing commands (block 508a), text commands (block 508b), and table commands (block 508c). The assumption in this example is that a user has a table selected, and thus all three of these blocks are relevant. A second type of object is a context pane. Context panes provide access to secondary tools for a context and are used to complete a specific task. The secondary tools are generally more specific or more advanced functionalities relating to the context block with which the pane is associated.

In the described embodiment, context blocks automatically appear and disappear based on the user's context and have two primary forms when visible: expanded and collapsed. Context blocks are expanded by default and collapse

007290" 9806560

only by manual action from the user. Context panes take up the entire context UI container 502 and focus the user on completing a task before they can continue with other commands in the application.

Context UI Container

The context UI container 502 is a collapsible vertical area that contains context blocks and context panes. The context UI container 502 can toggle between expanded and collapsed states as indicated above. When collapsed, the container is not visible at all. When expanded, the context UI container 502 is typically of a fixed size, e.g. 184 pixels wide, though it can grow wider to accommodate wider context blocks or context panes.

Changing the context UI container expansion state

In the described embodiment, the user can manually change the expansion state of the context UI container by clicking on a "Tools" button on a global toolbar in an application. This button can toggle container's expansion state. When the container is expanded, the button is visualized as toggled "on". When the container is collapsed, the button is visualized as toggled "off".

In addition, the context UI container 502 can be expanded programmatically. For example, if the user clicks on a command elsewhere in the application that requires that the container to be open, then the context UI container automatically opens. When the context UI container opens, the left border of the document being viewed by the application shifts over and the document's total width is decreased by the size of the context UI container 502.

and then sees to it that the proper context blocks are displayed. An exemplary expression-based system is described in more detail below in a section entitled “Expression Evaluation”.

Sizing and overflow issues

The context UI container 502 is of fixed size vertically, but the number and size of context blocks is not limited. Therefore, there may be situations in which an application does not have enough room to display all of the current context blocks. This is referred to as an “overflow case”. The overflow case occurs when there is not enough vertical room in the context UI container 502 to display all of the context blocks. One solution of the overflow case, in this particular example, is as follows: When an overflow occurs, the application can display a small scroll button at the bottom of the context block container. This button serves as an indicator that there are one or more context blocks scrolled out of the container. Clicking on the button once scrolls down by a predetermined number of pixels, e.g. 44 pixels, or to the end of the last context block, whichever is less. This, in turn, causes part or all of some context block to scroll off of the top of the context UI container 502. Accordingly, the context UI container 502 will also show a scroll button at its top when this occurs. Clicking on this top button will scroll up by a predetermined number of pixels, e.g. 44 pixels, or to the top of the first context block, whichever is less.

If there are no more context blocks or parts of context blocks scrolled out of the container in a certain direction (either up or down), then the corresponding respective scroll button will disappear. Since the scroll buttons take up space in the container 502, the calculation for when the scroll buttons should disappear

takes into account the additional room that would appear if the button were not there.

Application window resizing issues and the context UI container

The context UI container 502 is defined in the illustrated example to have a standard size horizontally and is sized to fit the application window vertically. The context UI container 502 responds to window/resolution resizing issues as follows: Vertically, the container resizes to fit into the space left over in the application frame from any other UI areas at its top or bottom. If the vertical space is not enough to hold all context blocks, then the overflow mechanism described above is invoked. Horizontally, the context UI container does not resize except to accommodate larger context blocks or context panes. Other than this case, the container only expands or collapses completely. The container does not resize horizontally due to the application window resizing. If the user resizes the window horizontally so that the window is narrower than the context UI container, the container will be clipped.

Context Blocks

In the illustrated example, context blocks are rectangular control containers that expose top-level commands for a given context. A context is anything that can be described by an “expression” in the application. Examples of expressions are given below. Typical contexts include: the type of document being currently viewed, the state that the document is currently in, for example, whether the document is in read-only (or browse) mode or is editable, and any objects that are currently selected in the document. Other contexts were listed above. Context blocks appear and disappear based on whether an expression that describes their

context is true or false, respectively. This is discussed in more detail in the “Expression Evaluation” section below.

Fig. 6 shows exemplary context block 508b apart from the context UI container 502 of Fig. 5. Context block 508b displays text formatting commands. Each block comprises a title bar area 600 and a controls area 602.

The title bar area 600 provides a location to label the entire context block, provides expand/collapse functionality, and also contains a button 604 that opens up the context block menu. The user can click anywhere on the title bar outside of the menu button to toggle the expansion state of the context block. On the right-hand side of the title bar area 600, button 604 can be clicked to bring up a menu that can contain links to context panes, as well as commands that execute immediately without invoking a context pane. The menu then closes after the user invokes a command from it.

The controls area 602 is the main area of the context block and exposes commands to the user. In the illustrated example, the controls area allows any arbitrary HTML code to be filled into it, so commands can be exposed in any way that HTML supports. Of course, other manners of defining commands other than HTML can be used. Context blocks are advantageously “modeless” meaning that any action taken by the user is immediately applied to the user’s selection in the document. This is advantageous over past methods because a user can experiment with different selections that are available through the context block and see their choices immediately effectuated in their document. In some cases, this eliminates the need for a costly (in terms of both space and time) “preview pane.” This does not, however, mean that the context blocks must always be modeless. For

Context panes can be accessed from the menus of their associated context blocks, through keyboard shortcuts, or from other UIs in the application, such as from a global toolbar. In the described embodiment, when a context pane is invoked, the entire content of the context UI container 502 (Fig. 5) is replaced with the context pane. Advantageously, context panes are typically modeless with respect to the document. This means that the user can continue to interact with their document while a context pane is open. For example, if a user wishes to use a strikethrough command repeatedly in a portion of text, the user can do this time after time by simply selecting the appropriate text and clicking on a strikethrough box in the context pane. In the past, a user would have had to traverse a menu structure for each separate invocation of the strikethrough command.

Fig. 7 shows an exemplary context pane 700 that includes secondary format font commands. Context pane 700 includes a title bar 702 and a controls area 704 that contains individual commands. A context pane looks similar to a context block that fills up the entire context UI container. There are, however, a few differences in the described example. Here, context panes are not collapsible. That is, they are displayed in their entirety for a user during the course of a user's interaction with them. In addition, context panes have a standard way to be closed, e.g. a "Close" button or the equivalent at the bottom of the panel can be clicked on by the user.

In addition, whereas a user does not have to request a context block in order for it to appear, a user does, in most cases, request a context pane for it to appear. That is, context panes are not automatically displayed as a result of an expression evaluation. Rather, they are displayed based on some event or function call. There can be, however, some context panes that are expression based and are not

1 to enter a search query through a dialog box that typically pulls up multiple topics
2 that relate to their search. They must then navigate through the different multiple
3 topics to find the one in which they are interested. "Help" in these instances, is
4 typically delivered as a separate application, overlaying, obscuring, or displaying
5 outside of the user's application window. There is no awareness of the context of
6 the user's work. Here, however, the described approach is somewhat different.
7 First, the help function is contextually related to the current context pane. Thus,
8 the user only receives help information that is pertinent to their current context.
9 Additionally, because the help information is specifically tailored to the user's
10 current context, and because there is a dedicated space for the context blocks and
11 context panes, more thorough help information can be displayed in the container
12 UI than would normally be possible in present systems. In addition, the help
13 feature is rendered in a "modeless" fashion so that the user can continue working
14 in their document while the help menu is displayed. Further, it is worth noting
15 that the contextual help provided by the present example is tailored not only to the
16 user's context, but to the tasks and troubleshooting steps that are most likely to be
17 needed by the user in that context. For instance, if the user is correcting a
18 misspelled word in the document by using a context pane designed for that
19 purpose, the help pane associated with that context pane may contain information
20 about how to correct the misspelled word with one of the provided choices, how to
21 add the word in question to the system dictionary, and how to replace the word in
22 question with a different word altogether. Accordingly, the user is provided with
23 assistance in a much more efficient and informative way.

Stackability

Although only one context pane can be viewed at a time, it is possible for multiple panes to be stored in a stack. In this case, closing one context pane reveals the next context pane in the stack. There are a number of cases where context panes can get stacked on top of each other. The following constitute some exemplary cases:

- A context pane is open and then an error context pane is displayed
- A context pane is open and the user then opens up another context pane from a button on the global toolbar.
- A context pane has a button that opens another context pane.
- A context pane is open and the user hits an accelerator key that opens up another context pane

In each case above, the latter context pane opened goes to the top of the stack, while the previous context pane goes underneath it on the stack.

In addition, each document can have its own stack of context panes. If the user navigates away from a document and back to it, the document's stack of context panes persists and is redisplayed (though it is possible for members of a stack to be aged out).

Expression Evaluation

As described above, context blocks are automatically presented to the user depending on the user's current context. In the described embodiment, an expression-based method is used to ascertain which contexts blocks to present and when to present them.

One way of implementing an expression-based method is as follows. Each context block is associated with an expression that can evaluate to a predetermined

value. Each expression is essentially a defined condition that describes some aspect of a user's interaction with a document. As a user interacts with a document, the expressions, or at least portions of the expressions, are evaluated to ascertain whether they evaluate to the predetermined value. When one or more of the expressions evaluates to the predetermined value, the context block that is associated with that expression is displayed for the user.

Fig. 8 is a flow diagram that describes steps in a method in accordance with the described embodiment. The described method can be implemented in any suitable hardware, software, firmware or combination thereof. In the illustrated example, the method is implemented in software.

Step 800 associates a context-sensitive UI with a visibility expression. An exemplary context-sensitive UI is a context block as described above. In the described example, a table is used for the association and includes two columns, one of which is associated with a particular context block, the other of which is associated with the context-block's visibility expression. Fig. 9 shows an exemplary table 900 with columns 902 and 904. Column 902 contains entries associated with each context block, while column 904 contains so-called visibility expressions that are associated with each of the context blocks. In the illustrated example, two exemplary context blocks are shown in column 902 with their corresponding visibility expressions in column 904. For example, for the "Font Format" context block the visibility expression is "em & ts". The visibility expression is a Boolean expression that describes a condition in which the application is in "edit mode" (i.e. "em") with a portion of text having been selected (i.e. "ts"). For the "Table Commands" context block, the visibility expression is "em & ip=t + tbs" which translates to a condition in which the

1 application is in edit mode and an insertion point lies within a table (i.e. "ip=t"), or
2 a table has been selected (i.e. "tbs").

3 Step 802 determines whether a visibility expression has changed in value
4 because of a user's action. A user's action can typically change their context. A
5 user's context could be based upon any type of variable such as user selection,
6 insertion point, time of day, user's name, to name just a few. If the value of a
7 visibility expression has changed, then step 804 removes visible UIs (i.e. context
8 blocks) that are not applicable to the current context. Step 806 displays UIs that
9 previously were not visible but are applicable to the user's current context.

10 The visibility expressions can be evaluated in any suitable fashion. For
11 example, each time the user takes an action within a document, all of the
12 expressions in table 900 can be evaluated. A more desirable approach is as
13 follows:

14 Each of the expressions is represented in a data structure known as a "tree".
15 Fig. 10 shows exemplary tree structures for the two illustrated visibility
16 expressions of Fig. 9. Here, the top node of each tree comprises an operation. In
17 the present case, the operation happens to be an "AND" operation for each
18 expression. Each top node has one or more children nodes that can either be
19 operands or operations. In the case of the font format context block expression,
20 each of the children nodes 1002, 1004 is an operand (i.e. "edit mode" and "text
21 selected" respectively). For the table commands context block expression, child
22 node 1008 is an operand (i.e. "edit mode") and child node 1010 is an operation
23 (i.e. an "OR" operation). In turn, node 1010 has two operand nodes 1012, 1014
24 (i.e. "insertion point = table", and "table selected" respectively).
25

Each tree structure can evaluate to either "TRUE" or "FALSE". If the tree structure evaluates to "TRUE", then its corresponding context block is displayed. If the tree structure evaluates to "FALSE", or remains in a false state after some of the nodes have been evaluated, the context block is not displayed. The expression value of a tree, however, cannot change unless the value of its operands changes. For example, consider the font format tree structure. Assume that its current value is "FALSE" (indicated by the "F" adjacent node 1000). Assume also that its edit mode operand 1002 has a value of "TRUE" and its text selected operand 1004 has a value of "FALSE". In this case, the user is currently in edit mode but has not selected any text. In order for this tree structure to change in value, the value of its text selected operand 1004 must change from "FALSE" to "TRUE". This will only happen when a user has selected some text with their cursor. In accordance with the described embodiment, when the value of a child node changes, it generates a notification to its parent node that its value has changed. The parent node expression is then re-evaluated to ascertain whether its value has changed. If its value has changed, then if it has a parent node, it generates a notification that is sent to its parent node. This continues until either a parent node's expression does not change in value, or the top parent node's expression changes in value. If the latter is the case, a corresponding context block is either displayed or removed. If the former is the case, then a current state is maintained (i.e. if the context block was previously displayed, then it is still displayed; and if the context block was not previously displayed, then it is not displayed). Thus, in many cases, only a portion of the visibility expression will need to be evaluated.

As another example, consider the visibility expression for the table commands context block. Assume that the current state of the expression is as indicated in the table below:

Node	Value
AND	FALSE
Em	TRUE
OR	FALSE
ip=t	FALSE
tab sel	FALSE

In this example, the table commands context block is not being displayed because the top node 1006 has evaluated to "FALSE". The user is in edit mode and neither the insertion point is in a table nor has a table been selected. Assume now that the user selects a table with their cursor. In this case, the value associated with node 1014 is changed to "TRUE". Because this node changed in value, it generates a notification and sends the notification to its parent node 1010. Node 1010 is an OR expression whose value now re-evaluates to "TRUE". Because this node has changed in value, it generates a notification and sends the notification to its parent node 1006. Node 1006 is an AND expression that now evaluates to "TRUE". Since this is the top node and it now evaluates to "TRUE", the context block with which it is associated is now displayed for the user. This logically follows from the user's actions. That is, in order to change the value of node 1014, the user had to select a table. When the user selects the table, the table commands context block should automatically be displayed for the user. If and

1 when the user "unselects" the table, the value associated with node 1014 will
2 change and this change will be propagated up the tree in the form of notifications
3 until the top node 1006 is re-evaluated to "FALSE" and the context block is
4 removed.

5 Fig. 11 is a flow diagram that describes steps in an exemplary expression-
6 evaluation method in accordance with the above-described embodiment. The
7 described method can be implemented in any suitable hardware, software,
8 firmware, or combination thereof. In the illustrated example, the method is
9 implemented in software.

10 Step 1100 represents each expression as a tree structure having multiple
11 nodes. Exemplary expressions and tree structures are shown and described in
12 connection with Figs. 9 and 10. Step 1102 gets the first tree or tree structure and
13 step 1104 gets the first leaf node on the tree. In the Fig. 10 example, exemplary
14 leaf nodes are shown at 1002, 1004 for the font format tree, and 1008, 1012, and
15 1014 for the table commands tree. Step 1106 evaluates the node. Step 1108
16 determines whether the node value has changed. If the node value has not
17 changed, then step 1110 determines whether there are more nodes on the tree, and
18 if so, step 1112 gets the next node and returns to step 1106. If there are no more
19 nodes on the tree, step 1114 determines whether there are more trees. If there are
20 additional trees, step 1116 gets the next tree and returns to step 1104 to evaluate
21 the nodes of the tree. If there are no additional trees, then step 1114 returns to step
22 1102. Note, that the return of step 1114 can take place automatically to repeat the
23 above process, or the return can be effected by a user's context change.

24 If, at step 1108, the node value has changed, step 1118 determines whether
25 this node is the root node of the tree. If the node is the root node, then the method

1 branches to step 1126 to ascertain whether the value of the node is "TRUE" or
2 "FALSE". If the value is "FALSE, then step 1128 hides the context block that is
3 associated with that particular visibility expression. If, on the other hand, the
4 value is "TRUE", then step 1130 displays the context block that is associated with
5 that particular visibility expression. If, at step 1118, the node is not the root node,
6 step 1120 gets the parent node of that particular node and step 1122 evaluates the
7 parent node. If the node value changes (step 1124), then the method branches
8 back to step 1118. If, on the other hand, the node value does not change, then the
9 method branches to step 1110.

10 The above-described process is advantageous in that many times the
11 complete expressions that are associated with the context blocks need not be
12 evaluated. Many times, only portions of the expressions need to be evaluated. If a
13 particular portion of the expression has changed in value, then additional portions
14 of the expression can be evaluated. If particular portions of the expression have
15 not changed in value, then it is possible to terminate the expression-evaluation
16 process thereby saving processing overhead.

17 Note that a small delay function can be built into the system so that the
18 expression evaluation process is not initiated immediately when a user takes a
19 particular action. For example, the system might be programmed so that the
20 expression evaluation process is not initiated until a user has left their cursor in a
21 particular location for a definable amount of time. Such delay mechanisms will be
22 understood by those of skill in the art and are not discussed in detail any further.
23
24
25

Single Navigable Window Application

In accordance with one implementation, the context-sensitive context blocks and context panes can be employed in connection with a single application program having multiple different functionalities to which a user can navigate and accomplish multiple different tasks. As the user navigates to the different functionalities, their context inevitably changes. As their context changes, so too do the context blocks and context panes that are displayed for the user. An exemplary single application program with multiple different functionalities is described in the U.S. Patent Application entitled "Single Window Navigation Methods and Systems", incorporated by reference above.

In the exemplary single application program that is subject of the reference incorporated above, software provides a user interface (UI) that presents a user with a single navigable window that can be navigated from functionality to functionality by a user. The individual functionalities are desirably provided by a single application program the result of which is a highly integrated software product.

A user, through the use of various navigation instrumentalities can navigate between the functionalities and when doing so, the single window ensures that only one functionality is presented to a user at a time. In this described embodiment, one navigation instrumentality is provided in the form of a web browser-like navigation tool. The choice of a web browser-like navigation tool follows from concerns that navigation instrumentalities be of a type that is readily understood by most individuals familiar with computing environments. Thus, when a user first encounters the inventive navigable single window concept for the first time, they do not have to learn an unfamiliar navigation concept. Another

1 navigation instrumentality includes links to each of the multiple different
2 functionalities. These links can be clicked on by a user and the single navigable
3 window is automatically navigated to the selected functionality.

4 Fig. 12 shows but one exemplary user interface (UI) 1200 in accordance
5 with one described embodiment. It will be appreciated that other UIs could be
6 used to implement the inventive concepts described herein and that the illustrated
7 UI constitutes but one way of doing so. In the illustrated example, UI 1200
8 includes a navigation bar 1202, one or more command areas 1204, and a display or
9 document area 1206 that constitutes the single navigable window.

10 Navigation bar 1202 is located adjacent the top of display area 1206 and
11 contains browser-like navigation buttons 1208 in the form of a “back” button, a
12 “forward” button, a “stop” button and the like. The navigation bar can be located
13 anywhere on the UI. Its illustrated placement, however, is similar in appearance to
14 the placement of traditional web browsing navigation features. In addition to the
15 navigation buttons 1208, the navigation bar 1202 also includes links 1210 to the
16 different functionalities that can be accessed by the user. In the illustrated
17 example, links to three exemplary functionalities (i.e. functionality 1, functionality
18 2, and functionality 3) are shown. These functionalities are typically different
19 functionalities that can enable a user to complete different respective tasks.
20 Examples of different tasks are given below in more detail. These functionalities
21 are all provided within the context of a single application. To access a particular
22 functionality, a user simply clicks on one of the links and a window that pertains
23 to the selected functionality is immediately presented in the display area 1206.

24 Command areas 1204 are located adjacent the top and left side of the
25 display area 1206. The command area(s) can, however, be located in any suitable

1 location. The command areas provide commands that are both global in nature
2 and specific to the particular context the user has selected. For example, some
3 commands such as “search” and “help” might be considered as global in nature
4 since they can find use in many contexts. Other commands, such as “text bold” or
5 “reply to all” are more specific to the particular context that the user has selected.
6 For the “text bold” command, the user’s context may likely be a word processing
7 context, while the “reply to all” command may likely be employed in an email
8 context.

10 **Example**

11 As an example of the single navigable window provided by a single
12 application consider Figs. 13 and 14.

13 In this example, the multiple functionalities 1210 that can be navigated by a
14 user include a browser functionality (indicated by the home icon), a mail
15 functionality (indicated by the letter icon), a planner functionality (indicated by the
16 clock icon), a contacts functionality (indicated by the people icon), a documents
17 functionality (indicated by the folder icon), and a links functionality (indicated by
18 the world icon). These functionalities are so-called “document-centric”
19 functionalities because they all relate in some way to a document that a user
20 interacts with, e.g. a Web page document, an email document, a calendar
21 document, etc.

22 Fig. 13 shows an example of a display that is rendered in the display area
23 1206 when a user clicks on the link to the browser functionality. By clicking on
24 the link (i.e. the home icon) to the browser functionality, single application
25 program software executing on the user’s computer executes to implement a

Fig. 14 shows an example of a display that is rendered in the display area 1206 when the user clicks on the link to the mail functionality (i.e. the folder icon). By clicking on this link, single application program software executing on the user's computer executes to implement the mail functionality. In this example, the mail functionality displays a user's inbox with messages that have been received by the user. Notice that the leftmost command area has been minimized by the user and that command area 1204 adjacent the top of the display area 1206 contains commands that are specific to the user's current context, e.g. "New" for generating a new email message, "Reply" for replying to an email message, "Reply to All" for replying to all recipients of an email message and the like. When the user's context within this functionality changes in a way that requires

1 one or more context blocks to be displayed, the context blocks will be
2 automatically displayed in the leftmost command area. For example, a user may
3 author an email message and desire to italicize a portion of text. Upon selecting a
4 portion of text, a text formatting context block will automatically appear for the
5 user to use. As another example, assume that a user incorporates a table into their
6 email message, if they then move the cursor inside of the table, the table
7 formatting context block will automatically appear in the leftmost command area.

8 Although not specifically illustrated, the user could have displays for the
9 planner, contacts, documents, and links functionalities presented in the display
10 area 1206 by simply clicking on the links to these specific functionalities. When
11 so displayed, context blocks that are associated with the user's context in these
12 particular functionalities will be automatically displayed in accordance with the
13 user's particular context. The navigation bar 1208 provides the user with the
14 ability to navigate through these different functionalities in a browser-like manner.

15 It is important to note that the above example constitutes but one exemplary
16 way in which multiple different functionalities and context blocks can be
17 presented to a user within the construct of a navigable structure. It should be
18 understood that the specifically illustrated functionalities (i.e. browser, mail,
19 planner etc.) constitute specific examples of different functionalities that are
20 capable of being incorporated into the single application program that provides the
21 navigable window. Accordingly, other different functionalities can be employed.
22 This aspect is discussed in more detail in the section entitled "Extensible
23 Functionalities" below. It should also be noted that various context panes are
24 associated with the individual context blocks that form the basis of this example.
25

1 The context panes have not specifically been described in this example because
2 they were explained above.

3 * Fig. 15 is a flow diagram that describes steps in a method in accordance
4 with the described embodiment. The illustrated method can be implemented in
5 any suitable hardware, software, firmware, or combination thereof. In the
6 illustrated example, the method is implemented in software.

7 Step 1500 provides a single application program with multiple different
8 functionalities. The functionalities, as pointed out above, are advantageously
9 different so as to enable a user to accomplish different tasks. One specific non-
10 limiting example of different functionalities was given above in the context of
11 document-centric functionalities that enable a user to make use of browser, mail,
12 planner, contacts, documents, and links functionalities. Step 1500 can be
13 implemented by configuring a computing device, such as a user's computer, with
14 the single application program having the multiple different functionalities. This
15 step can also be implemented by providing a software platform in the form of a
16 generic single application shell that is extensible and adaptable to receive different
17 extensions or software modules that embody various different functionalities, as
18 described in various patent applications incorporated by reference above. These
19 different extensions are then presented to the user in the context of the single
20 application having the multiple different functionalities.

21 These extensions can be delivered to the platform in any suitable way and
22 through any suitable delivery mechanism. For example, one way of delivering the
23 various extensions or functionalities is to deliver them via a network such as an
24 Intranet or the Internet. Regardless of the manner in which the single application
25 is provided, step 1502 presents a user interface (UI) with a single window and

1 links to the multiple different functionalities. The UI can also advantageously
2 include navigation instrumentalities that enable a user to navigate between the
3 different functionalities in a browser-like manner. Figs. 13-14 give specific
4 examples of an exemplary UI that can be used in accordance with the described
5 embodiment. Step 1504 ascertains whether a user has selected a particular link to
6 a functionality or whether the user has used one of the navigation instrumentalities
7 to navigate to a particular functionality. If a user has not done either, the method
8 branches back to step 1502. If, on the other hand, a user has selected a particular
9 link or used a navigation tool to navigate to a particular functionality, step 1506
10 presents a functionality-specific display within the single window. That is, the
11 single navigable window is navigated by the software to the selected functionality.
12 Specific examples of this were given above in connection with Figs. 13 and 14 in
13 which browsing and mail functionalities were respectively displayed within
14 display area 1206. In connection with presenting the functionality-specific display
15 in step 1506, step 1508 can present functionality-specific commands in a
16 command area of the UI. This is advantageously done automatically as a user
17 navigates from functionality to functionality. That is, as a user changes
18 functionalities, command sets that are specific to the user's current context or
19 functionality are automatically displayed in the command area. In connection with
20 this step, context blocks can be automatically displayed as described above. It will
21 also be appreciated that step 1508 includes the step of presenting various context
22 panes in response to the user selecting them as described above. Step 1508 then
23 branches back to step 1504 to ascertain whether the user has navigated to another
24 functionality.
25

0959086-062100

Application-level context panes and stacking

Application-level context panes are implemented with special behavioral characteristics with regards to the stacking of context panes. In this example, there are two types of context panes: those with affinity to a particular document, and those with no affinity to any document. A stack of context panes that have been opened is maintained. The stack is ordered so that the most recent pane is on the top of the stack. This stack does not contain any panes that have been explicitly closed by the user. The first pane in the stack that meets one of the following two criteria is displayed: (1) the pane has affinity to the current document, and (2) the pane has no affinity to any document. If no pane in the stack meets these criteria, then the context blocks are displayed. Note that this has the effect of hiding any pane that does not have affinity to the current document. This means that when navigation occurs, panes with affinity to the previous document are suppressed. Panes with affinity to the new document and those with no affinity to any document become candidates for display. They are considered as candidates because only the pane closest to the top of the stack is actually displayed.

Conclusion

The embodiments described above provide methods and systems that automatically present a user with commands that are specific to a task in which the user happens to be engaged. Advantageously, as the user's context changes within an application, the commands that are presented automatically change as well. In various implementations, the user can be given the opportunity to select additional context-sensitive commands for display. Overall, the methods and systems

1 advantageously enable a user to take advantage of many different commands
2 without requiring the user to know much about the application that they are using.
3 In one particular implementation, a single application comprises multiple
4 functionalities that enable a user to accomplish different tasks. These multiple
5 functionalities are presented in the context of a single window that is navigable by
6 a user between the different functionalities. Advantageously, navigation
7 instrumentalities are provided that are, in some instances, browser-like in
8 appearance and allow the user to navigate between the application-provided
9 functionalities in a browser-like manner. Functionality-specific commands can be
10 automatically presented to the user when they navigate to a particular
11 functionality. The functionality-specific commands are presented, in the
12 illustrated example, in the form of context blocks and content panes as described
13 above. One aspect of the single navigable window application is that the
14 application can serve as a basis for an extensible platform that can be configured
15 with different functionalities. Software modules incorporating the different
16 functionalities, as well as appropriate command sets that are displayable in the
17 context blocks and panes, can be desirably included in the software modules.
18 When the modules are plugged into the platform, a set of extensible functions is
19 provided. Each of the extensible functions can have their own set of unique
20 context blocks and panes that can be automatically displayed in a manner that is
21 defined by the software developer of the module.

22 Although the invention has been described in language specific to structural
23 features and/or methodological steps, it is to be understood that the invention
24 defined in the appended claims is not necessarily limited to the specific features or
25

001290" 9806560

1 steps described. Rather, the specific features and steps are disclosed as preferred
2 forms of implementing the claimed invention.

3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25